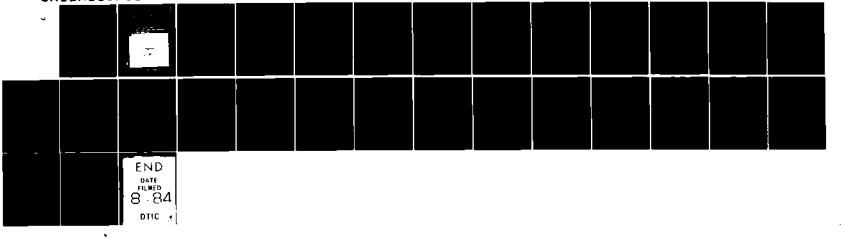
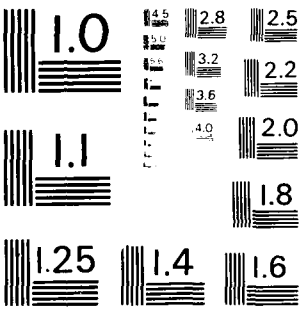
AD-A142 414 A MINIMUM AREA VLSI (VERY LARGE SCALE INTEGRATED ARCHITECTURE FOR O(LOGN)..(U) ILLINOIS UNIV AT URBANA APPLIED COMPUTATION THEORY GROUP G BILARDI ET AL. 1/1

UNCLASSIFIED NOV 83 ACT-45 N00014-79-C-0424 F/G 9/2 NL



END
DATE FILMED
8-84
DTIC



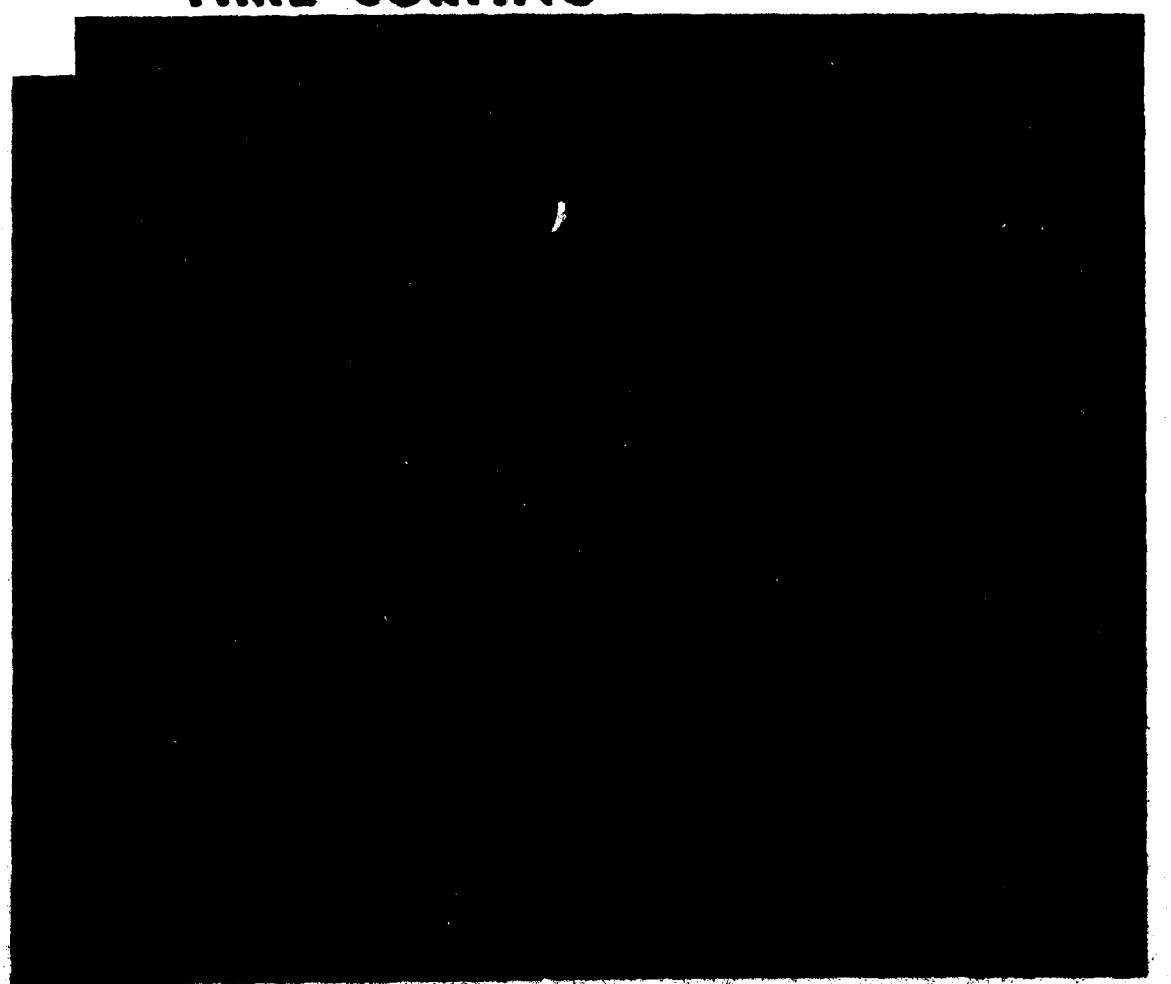
(12)

REPORT ACT-45 NOVEMBER 1983

COORDINATED SCIENCE LABORATORY

APPLIED COMPUTATION THEORY GROUP

A MINIMUM AREA VLSI ARCHITECTURE FOR O(LOGN) TIME SORTING



UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) R-1006; UILU-ENG 83-2227; ACT-45	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Laboratory, Univ. of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State and ZIP Code) 800 N. Quincy Street Arlington, VA
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Services Electronics Program 8b. OFFICE SYMBOL (If applicable) N/A	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Contract N00014-79-C-0424
8c. ADDRESS (City, State and ZIP Code) 800 N. Quincy Street Arlington, VA	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: N/A PROJECT NO.: N/A TASK NO.: N/A WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) A Minimum Area VLSI Architecture for O(logn) Time Sorting	
12. PERSONAL AUTHOR(S) Bilardi, G. and Preparata, F. P.	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) November 1983 15. PAGE COUNT 26
16. SUPPLEMENTARY NOTATION N/A	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) VLSI complexity, area-time trade-off, combination sorting, bitonic merging, cube-connected-cycles, mesh, orthogonal trees, optimal algorithms, parallel computation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A generalization of a known class of parallel sorting algorithms is presented, together with a new architecture to execute them. A VLSI implementation is also proposed, and its area-time performance is discussed. It is shown that an algorithm in the class is executable in O(logn) time by a chip occupying $O(n^2)$ area. The design is a typical instance of a "hybrid architecture", resulting from the combination of well-known VLSI arrays as the orthogonal-trees and the cube-connected-cycles; it is also the first known to meet the $AT^2 = \Omega(n^2 \log^2 n)$ lower bound for sorters of n words of length $(1+\epsilon)\log(n) - 0$), and working in minimum O(logn) time.

Fasil 9

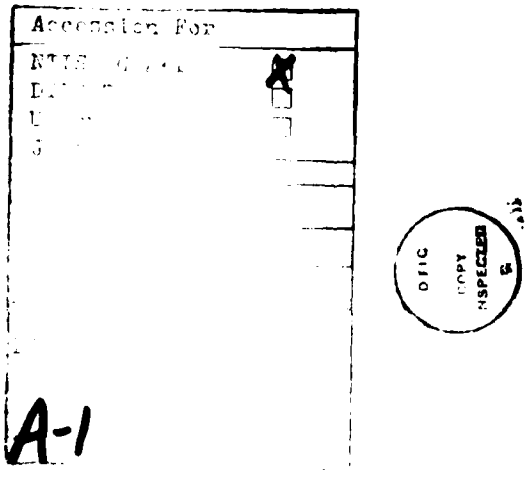
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code) 22c. OFFICE SYMBOL NONE

DD FORM 1473, 83 APR

A MINIMUM AREA VLSI ARCHITECTURE FOR O(LOGN) TIME SORTING

G. Bilardi and F. P. Preparata
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 West Springfield Avenue
Urbana, IL 61801

Abstract: A generalization of a known class of parallel sorting algorithms is presented, together with a new architecture to execute them. A VLSI implementation is also proposed, and its area-time performance is discussed. It is shown that an algorithm in the class is executable in O(logn) time by a chip occupying $O(n^2)$ area. The design is a typical instance of a "hybrid architecture", resulting from the combination of well-known VLSI arrays as the orthogonal-trees and the cube-connected-cycles; it is also the first known to meet the $AT^2 = \Omega(n^2 \log^2 n)$ lower bound for sorters of n words of length $(1+\varepsilon)\log n (\varepsilon > 0)$, and working in minimum O(logn) time.



This work was supported in part by the Joint Services Electronics Program under Contract N00014-79-C-0424 and by an IBM predoctoral Fellowship.

1. Introduction

Sorting is one of the most widely studied problems from the computational point of view, and many algorithms have been proposed for its solution. Since the possibility of parallel computation has been considered, several parallel schemes have also been proposed for sorting. Different models for parallel computers are possible, and several have been considered in the literature during the past years. Recently, the advent of the Very Large Scale Integrated circuits (VLSI) has motivated the definition of a new model of computation that aims at capturing the essential features of the new technology. Obviously, sorting has been one of the first problems studied in the VLSI environment, and several results are already available. In particular Thompson [1] gives a survey of thirteen algorithms for sorting and discusses their performance in terms of the chip area A and of the time T that elapses between beginning and completion of a computation. Indeed, area and time are natural measures of complexity for VLSI circuits, reflecting production cost and incremental cost respectively.

A theoretical argument, due to Thompson [2], shows that any sorter of n terms, with wordlength $q = (1+\varepsilon)\log n$, with $\varepsilon > 0$, must satisfy the relationship $AT^2 = \Omega(n^2 \log^2 n)$. The argument is based on the facts that any chip that sorts must support a flow of $\Phi = \Omega(n \log n)$ bits through a suitable bisection, and that $AT^2 = \Omega(\Phi^2)$. This lower bound holds in a suitable VLSI model of computation whose basic assumptions are that the chip is synchronous (transmission time is independent of wire length) and semellective-unilocal (input data are read only once, at prespecified input ports). A word-local restriction is also assumed for the input format (all the bits of the same word enter the circuit at the same point).

In a previous paper [3] we have shown that optimal VLSI sorters can indeed be constructed for all computation times $T \in [\Omega(\log^3 n), O(\sqrt{n \log n})]$. These sorters are based on a new architecture, the Pleated-Cube-Connected-Cycles (PCCC), and execute bitonic sorting [4].

In this paper we concentrate on "very fast" sorting, i.e., the class of VLSI sorting algorithms whose running time is $T = \theta(\log n)$. So far only one VLSI design is known to achieve $\theta(\log n)$ computation time: it is based on the orthogonal trees architecture [5], [6] and implements an algorithm due to Muller and Preparata [7].(1) The optimal layout of the orthogonal trees has area $A = O(n^2 \log^2 n)$ [6], while the lower bound yields $A = \Omega(n^2)$ for $T = O(\log n)$. On the other hand, a closer analysis of the algorithm shows that the information flow φ is $O(n \log n)$, so that the gap between upper and lower bounds is not due to a gap between actual flow and a flow-based lower bound, but it is due to the fact that the length of the layout bisection of the orthogonal trees is $O(\log n)$ times as large as the graph bisection.

We will show in this paper that not only the lower bound on the flow, but also the one on the AT^2 measure is tight, by exhibiting a new architecture capable of sorting in $A = O(n^2)$ and time $T = O(\log n)$.

The rather complex network is a typical instance of the "hybrid architecture", resulting from the careful interplay of more standard VLSI networks, as the cube-connected-cycles machine, the mesh-connected machine, and the binary-tree machine. The implemented algorithm is of the type first introduced by Preparata [8], although the recursion strategy has been modified to optimize the network area.

A slight modification of one of the building blocks of our sorter turns out to be an interesting network in its own right. It is called the <u>mesh of CCC's</u>, and is a powerful emulator of the binary cube, matching the performance of both the CCC and the PCCC machines.

A suitable combination of one $O(\log n)$ sorter and one mesh of CCC's of proper size will allow us to construct an AT^2-optimal sorter for any computation time $T \in [\Omega(\log n), O(\log^3 n)]$.

---

(1) Subsequent to the research leading to this paper, we learned of the construction of Aitai, Komlos, and Szemeredi [13], which also achieves $\theta(\log n)$ time; in addition we have devised a VLSI implementation of their

Thus we are able to conclude that optimal $AT^2 = \theta(n^2 \log^2 n)$ sorting is achievable in the entire "meaningful" range of computation times $T \in [\Omega(\log n), O(\sqrt{n \log n})]$. (Simple fan-in arguments show that $\Omega(\log n)$ is a lower bound for the computation time, and $A = \Omega(n \log n)$ is an immediate consequence of the semellective assumption, so that computation times slower that $\theta(\sqrt{n \log n})$ cannot result in smaller area.)

In Section 2 we introduce a general framework for sorting algorithms, called COMBINE-SORT, which is based on an operation, COMBINATION, generalizing the operation of MERGING from two to several sequences. Section 3 and 4 describe an architecture (COMBINER) and an algorithm for COMBINATION, respectively. The combiner network so obtained is then used in Section 5 as a building block for a general class of COMBINATION-sorters. One of these sorters is shown to have optimal area $A = \theta(n^2)$, for $T = \theta(\log n)$ computation time. Finally, in Section 6, we discuss the area-time trade-off for sorting, and show that optimal sorters can be constructed for any computation within the above range.

2. A Class of Parallel Sorting Algorithms

Several sorting algorithms can be viewed as particular cases of a rather general scheme, which we now describe.

We call COMBINATION the operation that produces from m sorted sequences of t elements each one sorted sequence of mt elements. A network implementing this operation is called an (m,t)-COMBINER. When m = 2, COMBINATION reduces to merging.

A parallel algorithm for the (m,t)-COMBINER has been introduced in [8], and is based on the following idea. The m input sequences $S_0,\ldots,S_{m-1}$ are pairwise merged to compute for each $i,j \in \{0,1,\ldots,m-1\}$, and each $\ell \in \{0,1,\ldots,t-1\}$, the number $C_{ij}(\ell)$ of elements of sequence S_j that are less than the ℓ-th element of sequence S_i. $C_{ij}(\ell)$ is readily obtained as the difference of the ranks of this element in the merge of S_i and S_j and in S_i. By summing the $C_{ij}(\ell)$'s over j we then obtain the rank of the ℓ-th element of S_i in the output sequence of the COMBINER; thus, to complete the operation, we simply need to store each element in the position specified by its rank. The primitive operation of the scheme -- the merging of two sequences -- can be done, for example by Batcher's bitonic merger [4].

Given $n = m_1 m_2 \ldots m_d$ elements, we can sort them in d stages according to the following scheme that we call COMBINE-SORT.

At stage 1 we perform n/m_1 combination operations, each on m_1 sequences of 1 element each. At stage 2 we perform $n/m_1 m_2$ combinations, each on m_2 sequences of m_1 elements each, and at stage i we perform $n/m_1 \ldots m_i$ combination, each on m_i sequences of length $m_1 \ldots m_{i-1}$. Finally, at stage d we combine m_d sequences of length n/m_d into one sequence of length n, which is the output of the COMBINE-SORT scheme.

A diagrammatic illustration of the scheme is given in Figure 1 in the form of a rooted tree. Each node of this tree is a suitable combiner. An (m_i,t_{i-1})-COMBINER, $1 \leq i \leq d$, performs the combination of m_i (sorted) sequences of length t_{i-1}; here $t_0 = 1$ and $t_{i-1} \triangleq m_1 m_2 \ldots m_{i-1}$ for $i > 1$. Note that each level of the tree corresponds to a stage of the combination scheme, and that there are $n_i \triangleq n/t_i$ nodes at level i, $1 \leq i \leq d$.

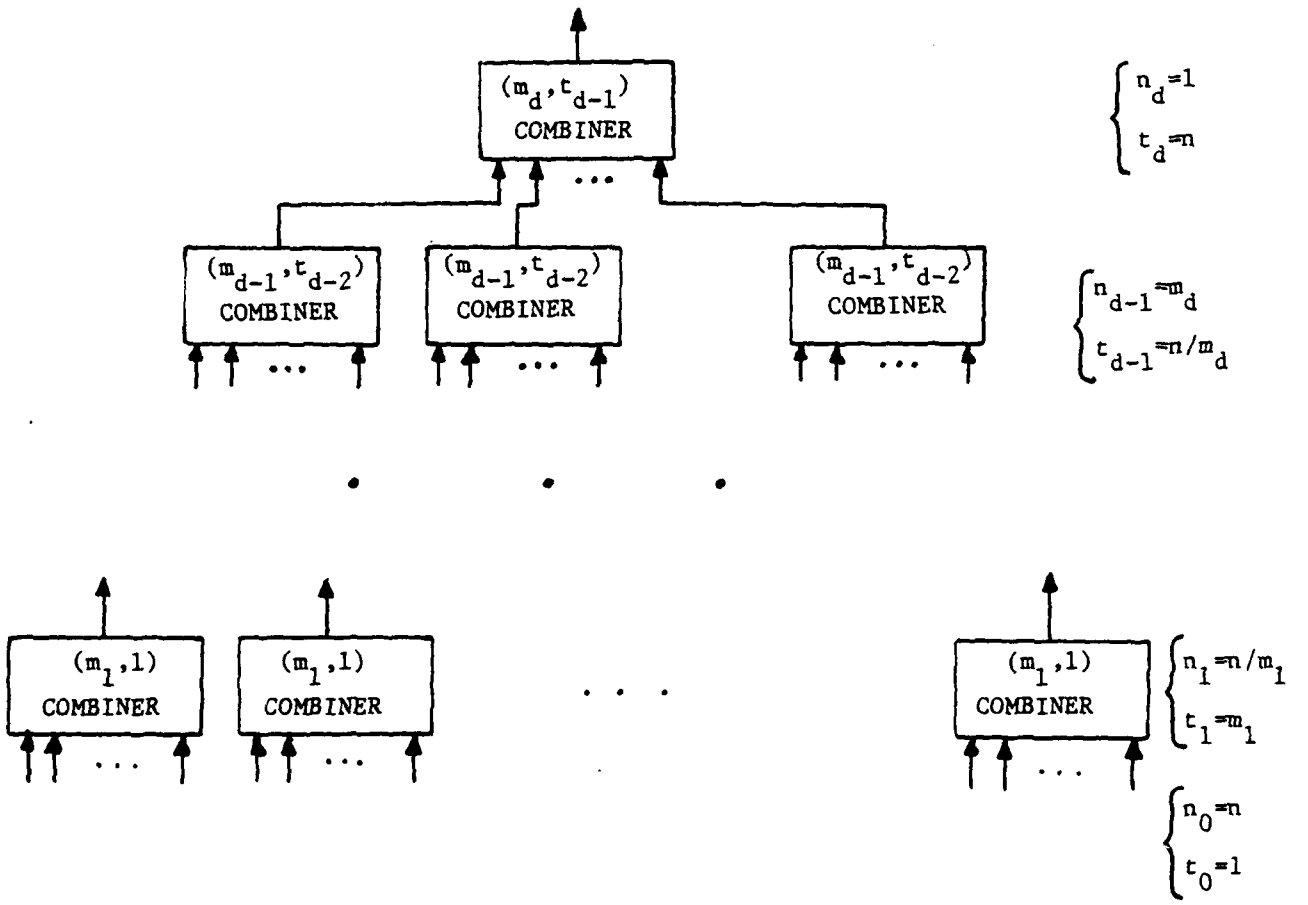


Figure 1. Diagram of COMBINE-SORT scheme.

Several known sorting algorithms can be cast in the COMBINE-SORT scheme. Each algorithm is characterized by a particular factorization of $n = m_1 \ldots m_d$ (note that the order of the factors is relevant here), and by the specification of how the combination is to be performed. In particular if we use the COMBINER based on [8] we have the following cases.

(i) When $n = 2^d$, and $m_1 = m_2 = \ldots = m_d = 2$, then COMBINE-SORT reduces to the usual MERGE-SORT.

(ii) When $d = 1$, and $m_1 = n$, the COMBINE-SORT reduces to only one $(n,1)$-COMBINER, which is essentially the sorting network described in [7].

(iii) When $d = \text{loglogn}/\log(1/(1-\alpha))$, and $m_{d-i} = n^{\alpha(1-\alpha)^i}$ with $0 < \alpha < 1$, we obtain the sorting schemes described in [8]. The sorting scheme corresponding to a given α can be described as follows. The n-input sequence is split into n^{α} (m_d in our terminology) sequences of $n^{(1-\alpha)}$ (t_{d-1} in our terminology) elements each. These sequences are sorted recursively, and then combined by an (m_d, t_{d-1})-COMBINER . The recursion stops when sequences of length 1 are obtained. We can obtain the values for d and $m_1, \ldots, m_d$ by a simple analysis of the unfolded recursive process.

In the following sections, we shall explore which other choices of d and $m_1, m_2, \ldots, m_d$ can be made to minimize the complexity of a VLSI implementation of COMBINE-SORT.

3. An (m,t)-COMBINER Network

In this section we propose a parallel architecture for an (m,t)-COMBINER, where $m = 2^{\mu}$ and $t = 2^{\tau}$ are powers of two. This architecture will accept as input m sorted sequences of t elements each,

$$S_i = (s_i(0), s_i(1), \ldots, s_i(t-1)) \qquad i = 0, 1, \ldots, m-1$$

and produce as output a single sorted sequence S, which is the combination of $S_0, \ldots, S_{m-1}$, and has $N = mt \triangleq 2^{\nu}$ elements,

$$S = (s(0), s(1), \ldots, s(N-1)).$$

The (m,t)-COMBINER will execute the algorithm based on pairwise merging as outlined in the preceding section. Its organization is illustrated in Figure 2. It consists of m^2 modules (each capable of merging two sequences of length t and of computing partial ranks), laid out as a square m × m mesh and indexed as M_{ij} (i,j = 0,1,...,m-1). The modules of each row are interconnected as the leaves of a binary tree of bandwidth t; so are the modules of each column. Thus, the combiner has the structure of the orthogonal-trees machines [5,6], whose leaves are merging modules. The interconnecting trees have the following functions:

(i) to "broadcast" a sequence to all units in which it must be merged with some other sequence;

(ii) to compute global ranks from partial ranks;

(iii) to rearrange the elements according to their ranks into the sorted sequence S.

We will now describe in some detail the merging modules and the interconnecting trees.

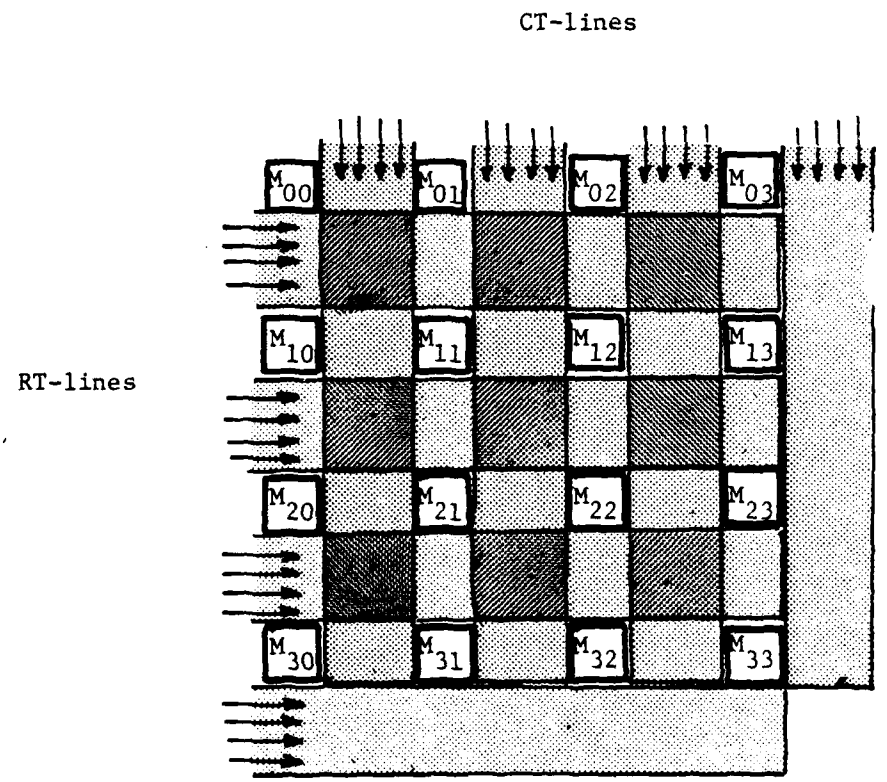


Figure 2. Overview of (m,t)-COMBINER, for m = 4.

3.1. Merging Modules

Merging module M_{ij} will merge sequences S_i and S_j and compute $C_{ij}(\ell)$, for $\ell = 0,\ldots,t-1$. We recall that $C_{ij}(\ell)$ is the number of elements of S_j that are less than (respectively less than or equal) $s_i(\ell)$ when $i \leq j$, (when $i > j$).

Each module is realized (Figure 3) as a cube-connected-cycle (CCC), interconnection of smaller processing elements, called micromodules (each micromodule has a bandwidth of 1 bit). Specifically, the merging module is a $(\tau+1,2^{\tau+1})$-CCC (i.e., it has $2^{\tau+1}$ cycles each of length $\tau+1$). We number the micromodules of M_{ij} as $M_{ij}(h,k)$, with $0 \leq h < \tau+1$ and $0 \leq k < 2^{\tau+1}$, so that the merging module may be thought of as a $(\tau+1) \times 2^{\tau+1}$ array (rows are numbered from bottom to top, columns from left to right). The columns of this array are connected as cycles with a link between $M_{ij}(h,k)$ and $M_{ij}(h,(k+1)\mathrm{mod}(\tau+1))$. The rows $0,1,\ldots,\tau$ are associated with the dimensions $E_0,E_1,\ldots,E_\tau$ of a $(\tau+1)$-dimensional binary cube [9], and there is a link between $M_{ij}(h,k_1)$ and $M_{ij}(h,k_2)$ if and only if the binary expansions of k_1 and k_2 differ exactly in the coefficient of 2^h.

The reader is referred to [10] for a detailed explanation of the CCC; he must also be warned that in this paper we will not use the CCC at its full capability, since we deploy a network with $2t(\log 2t)$ (rather than $2t$) micromodules to merge two sequences of length t. In other words, a $2^{\tau+1}$ binary cube is emulated by a $(\tau+1,2^{\tau+1})$-CCC.[2] When the $2^{\tau+1}$ items on which we operate have to be processed on the cube dimension E_h, we just need to guarantee that the items are in row h of the CCC. Thus, execution of the ASCEND and DESCEND paradigms, in which the dimensions are used in the sequence $(E_0,E_1,\ldots,E_\tau)$, and $(E_\tau,E_{\tau-1},\ldots,E_0)$ respectively, is quite straightforward.

The layout of a $(3,2^3)$-CCC in Figure 3 shows two sets of 4 input lines (denoted, respectively, as RT- and CT-lines) each carrying one of the two 4-element sequences to be merged.

[2] For this reason the number of micromodules in a cycle is not constrained to be a power of 2.

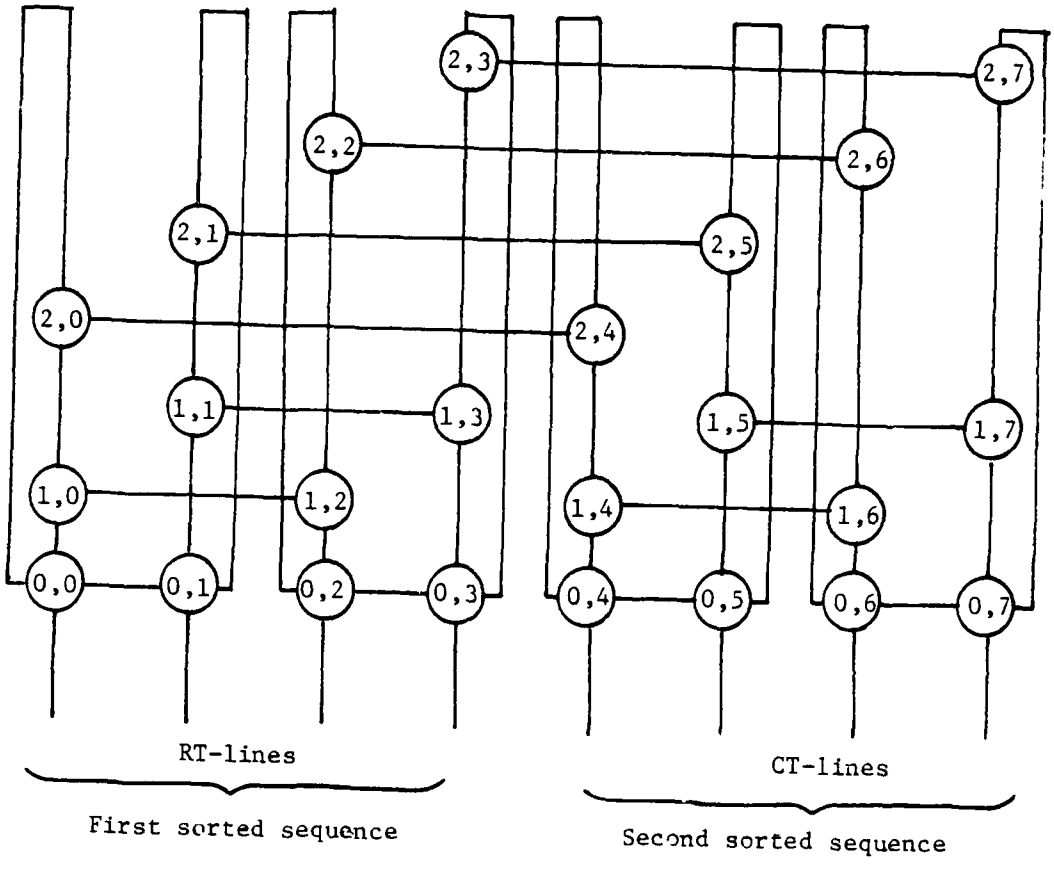


Figure 3. Merging unit M_{ij} realized by a $(3,2^3)$-CCC, used to merge two sequences with four elements each.

Recalling from [10] that a CCC with N processing elements of constant area can be laid out in area $O(N^2/\log^2 N)$, we conclude that a merging module can be laid out in area $O(A_0 t^2)$, where A_0 is the area of the micromodule. In the next section, in connection with the COMBINATION algorithm, we shall specify the functional capabilities of each micromodule, from which it will be clear that A_0 is constant, i.e., independent of the problem size.

3.2. Interconnecting Trees

As indicated earlier, the merging modules are interconnected by two families of $N = mt$ complete binary trees with $m = 2^{\mu}$ leaves and bandwidth 1. We will refer to these families as the row trees and column trees.

The lines of the row trees and the column trees are respectively labelled $RT_i(\ell)$ and $CT_i(\ell)$, $i = 0,\ldots,m-1$; $\ell = 0,\ldots,t-1$. The trees and the merging modules are connected through a small interface, whose structure will be fully specified in connection with the description of the COMBINATION algorithm in the next section. At this point we just say that the leaves of $RT_i(\ell)$ are, from left to right, connected to the CCC micromodules $M_{i0}(0,\ell), M_{i1}(0,\ell),\ldots,M_{i,m-1}(0,\ell)$; the leaves of $CT_j(\ell)$ are connected to the CCC micromodules $M_{0j}(0,t-1+\ell), M_{1j}(0,t-1+\ell),\ldots,M_{m-1,j}(0,t-1+\ell)$; in other words, the row trees and the column trees are respectively connected to the RT and the CT lines of the merging modules. The connection between each leaf of a tree and the corresponding CCC micromodule is realized through a buffer register of the appropriate size (adequate to store one element to be sorted). The situation is illustrated in Figure 4.

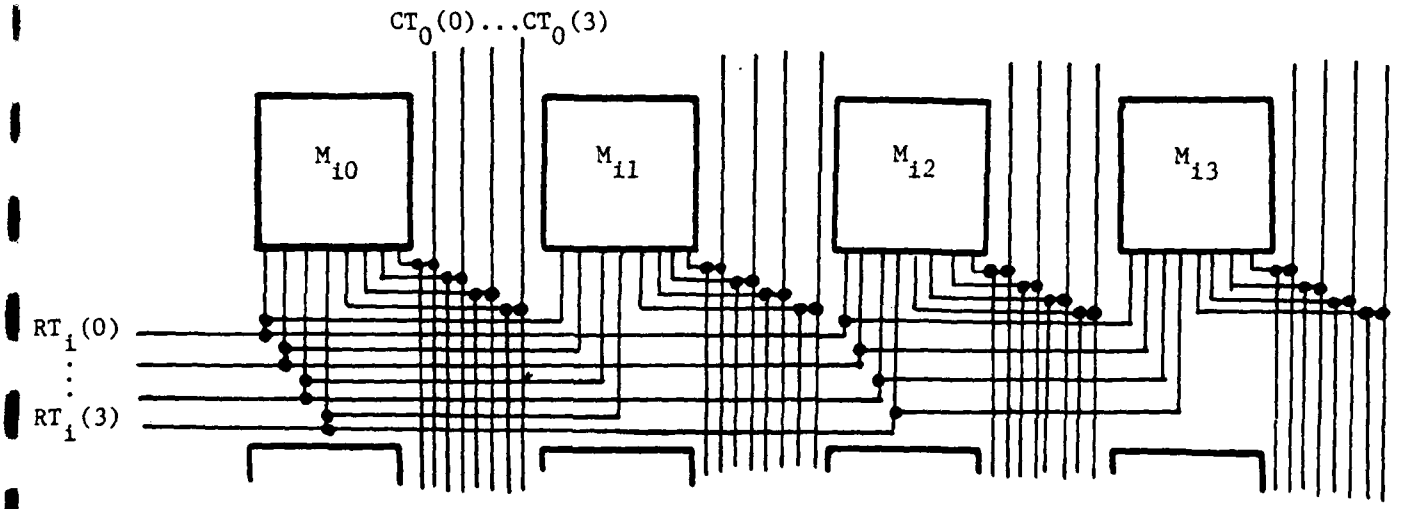


Figure 4. Interconnection of modules and trees.

4. The COMBINATION Algorithm

We now describe how the sorting algorithm of [8], based on pairwise merging, can be executed on the architecture introduced in Section 3. This analysis will elucidate the structure of the CCC micromodules. We recall that the inputs are $m = 2^{\mu}$ sorted sequences of $t = 2^{\tau}$ elements each, with $N = 2^{\nu} = mt$. For convenience we split the algorithm into several phases.

(A) Input of Data and Broadcasting to Merging Modules

Element $s_i(\ell)$ is input at the root of tree $RT_i(\ell)$, and is then broadcast to all leaves of the tree. At this point, the left half of row(0) in module M_{ij} contains the sequence S_i. To fill the right halves of row(0) of all modules, we proceed as follows. First, in each "diagonal" module M_{ii} the sequence S_i is copied in the second half of row(0). (This can be done by using the connection of row(τ) between the left and the right half of the machine.) Next, from micromodule $M_{jj}(0,t-1+\ell)$, which is a leaf of $CT_j(\ell)$, element $s_j(\ell)$ is broadcast (through the root) to all the other leaves of the same tree. At this point, the merging module M_{ij} contains S_i and S_j in the 0-th row and merging can begin.

(B) Merging and Partial Rank Computation

Merging can be executed by resorting to the bitonic algorithm, which complies with the DESCEND paradigm of the binary cube (see [10]). However, in order to execute bitonic merging, we first need to reverse the order of S_j. This is accomplished by an ASCEND algorithm in which columns t to 2t-1 of each M_{ij} exchange their data at dimensions $E_0,\ldots,E_{\tau-1}$, while columns 0 to t-1, remain idle. All the columns are idle at dimension E_{τ}.

Now the data are ready for bitonic merging. At each dimension, $E_{\tau},E_{\tau-1},\ldots,E_1,E_0$, pairs of elements are compared and exchanged, if necessary, to place the smaller of two in the column with the smaller number. Each

processor (micromodule) of the merging module is equipped with a serial comparator that reads the inputs starting from the most significant bit. As long as the two inputs agree, they are transmitted to the next processor in the same column. As soon as a bit discrepancy is detected, a switch is set and, from now on, the remaining substrings of the operands will follow a fixed path, respectively independently of their value. It is then easy to see how the computation through the rows of the CCC can be naturally pipelined to achieve a computation time of $O(r+q)$, where q is the length of the input words. At the end of merging, the result resides in row(0) of the CCC, and the element in $M_{ij}(0,\ell)$, $0 \leq i \leq 2t-1$, has rank ℓ in MERGE(S_i,S_j). Now we want to transmit the ranks of $s_i(0),\ldots,s_i(t-1)$ to processors $M_{ij}(0,0),\ldots,M_{ij}(0,t-1)$, respectively. This is accomplished by retracing backwards the path traversed by each element $s_i(j)$, and is easily done if each $M_{ij}(\ell,k)$ keeps track of whether it exchanged or not the operands during the merging process. So, all we have to do is to run the machine backwards, with an ASCEND algorithm, which applies to the ranks the inverse of the permutation that merged the elements. At the end of this phase, processor $M_{ij}(0,\ell)$, $0 \leq \ell \leq t-1$, stores the number of elements in MERGE(S_i,S_j) that are less than $s_i(\ell)$. If from this number we subtract ℓ we obtain $C_{ij}(\ell)$, number of elements of S_j which are less than $s_i(\ell)$. We call the C_{ij}'s <u>partial ranks</u> because from them we can compute the rank of each $s_i(\ell)$ in the sorted sequence S as $C_i(\ell) = \sum_{j=0}^{m-1} C_{ij}(\ell)$.

(C) <u>Total Rank Computation</u>

It is immediate to see that at the end of phase B the partial ranks $C_{i0}(\ell),C_{i1}(\ell),\ldots,C_{i,m-1}(\ell)$ of $s_i(\ell)$ are available exactly at the leaves of row tree $RT_i(\ell)$. By having in each internal node of the tree a full adder with a 1-bit delay feedback on the carry, we can then obtain at the root

of RT_i the sum $C_i(\ell)$ of the values stored at the leaves. The nodes work as serial adders and the tree is used in a pipelined fashion, so that the time required is $O(\mu+\tau)$, where $\mu = \log m$ is the depth of the tree, and $\tau+1$ is the wordlength of the operands (note that $C_{ij}(\ell) \leq 2^{\tau}$). Within the same order of time, we can subsequently broadcast $C_i(\ell)$ from the root to the leaves. (Indeed $C_i(\ell) < 2^{\tau+\mu}$, so it can be expressed by $\tau+\mu$ bits.)

(D) Sorting Permutation and Output of Data

We want to output the elements $s(0),\ldots,s(N-1)$ of the sorted sequence from the roots of the column trees, and, specifically, we want the root of $CT_j(\ell)$ to output element $s(j2^{\tau}+\ell)$. This corresponds to a natural left-to-right order of the column trees as they appear in the layout of Figure 2.

Considering a generic element $s_i(h)$ with rank $C_i(h)$, the binary spellings of the integers j and ℓ so that $s_i(h)$ will emerge from the root of column tree $CT_j(\ell)$ are readily obtained by taking the μ most significant bits and the τ less significant bits of the rank $C_i(h)$ to represent ℓ and j, respectively. Thus, as a first step, we "activate" in M_{ij} the elements of sequence S_i that have to emerge from trees CT_j's, and "inhibit" all other elements. The active elements are those whose rank $C_i(h)$ has the μ most significant bits agreeing with the column number j of the merging module. Next, we rearrange the active elements in M_{ij} so that $s_i(h)$ is sent to $M_{ij}(0,\ell)$, with $\ell = C_i(h) \bmod t$.

This operation is essentially a permutation of the active (and non-active) elements, and can be done by using the CCC as an emulator of the Benes-network. The setting of the switches, although nontrivial, is greatly simplified with respect to the general case by the fact that the active elements do not change their relative order. The desired rearrangement can be done by using the idea of concentration introduced in [11], and expansion, which could be viewed as the inverse of concentration. If k elements are active in a

given module, they are first sent to the k leftmost columns of the CCC (concentration), and then routed to the destination columns (expansion). A straightforward adaptation of the algorithm that is proposed in [11] for concentration in the cube-machine shows that an ASCEND and a DESCEND phase is all that is required to rearrange data on our CCC. Some bits required to set the switches must be precomputed. This task could be performed by the CCC, or (to keep the micromodule structure as simple as possible), the task can be assigned to a binary tree of full adders whose leaves would be contained in the interface between the CCC and the row-trees.

During the entire rearrangement task, computation takes place only in the left-half of the CCC without using dimension E_τ. We then transfer each active element from $M_{ij}(0,\ell)$ to $M_{ij}(0,t-1+\ell)$, with a straightforward use of dimension E_τ.

At this point element $s(j2^\tau+\ell)$ is in $M_{ij}(0,t-1+\ell)$, (where the value of i is determined by the input sequence to which $s(j2^\tau+\ell)$ originally belongs to), and is ready to be transmitted to the root of $CT_j(\ell)$ where it is output.

4.1. Performance Analysis and Modification of the Network

The entire machine, even when not explicitly said, is intended to work in bit serial mode. Both the CCC's and the trees work in a pipeline fashion. Thus any operation takes essentially time proportional to the sum of the operand length and the pipe depth. For the CCC's the depth is $\tau+1$. The operands to be handled have length q when they are input words or $\tau+1$ when they are partial ranks. Since a constant number of ASCEND and DESCEND algorithms are executed, we conclude that $O(\tau+q)$ total time is spent in the CCC's. For the trees the depth is $\mu+1$. The operands to be handled have length q when they are input words, or $\tau+\mu$ when they are total ranks. Since a constant number of

fan-in and fan-out algorithms are executed, we conclude that $O(\tau+\mu+q)$ total time is spent in the trees. Thus the time spent in the interconnecting trees dominates that spent in the CCC's, and we reach the conclusion that the $(2^{\mu},2^{\tau})$-COMBINER of elements of q bits works in time $T = O(\tau+\mu+q)$.

So far, all the parameters τ, μ, and q have been regarded as independent of each other. We now make an interesting observation. When $q = \Omega(2^{\mu})$, then $T = O(\tau+q)$. In this case the time performance of the trees is not substantially degraded if we realize them as comb-trees, rather than as complete binary trees. The depth will go from μ to 2^{μ}, but this is tolerable in time since $2^{\mu} = O(q)$. On the other hand comb-trees can be laid out in constant rather than logarithmic width, thus yielding a saving in area. The modified $(2^{\mu},2^{\tau})$-COMBINER of words of length $q = \Omega(2^{\mu})$ has then $T = O(\tau+q)=O(\log N+q)$ and $A = O(2^{2(\tau+\mu)}) = O(N^2)$.

4.2. Summary of Symbols and Results for an (m,t)-COMBINER

Sizes: $m = 2^{\mu}$, $t = 2^{\tau}$, $N = mt$, q = wordlength

Input sequences:

$S_i = (s_i(0),s_i(1),\ldots,s_i(t-1))$ $i = 0,1,\ldots,m-1$.

Output sequence:

$S = (s(0),s(1),\ldots,s(N-1))$.

Merging modules: $(\tau+1,2^{\tau+1})$-CCC's

M_{ij}: $i,j = 0,1,\ldots,m-1$

$M_{ij}(h,k)$; $0 \leq h < \tau+1$, $0 \leq k < 2t$, micromodules of M_{ij}.

Row-trees and column-trees:

$RT_i(\ell)$, $CT_j(\ell)$: $0 \leq i,j \leq m-1$, $0 \leq \ell \leq t-1$.

Machine	Performance	
	A	T
Full tree version	$O(N^2\mu^2)$	$O(\tau+\mu+q)$
Comb-tree version $q = \Omega(m)$	$O(N^2)$	$O(\tau+q)$

5. An Architecture for COMBINATION-SORT

We shall now use the COMBINER developed in the two preceding sections to construct a general network for COMBINATION-SORT. As an intermediate step in the construction, we introduce a new operation called COALESCENCE. Given a collection of n elements, partitioned into n/t_{i-1} sorted subsequences each containing t_{i-1} elements, and given a multiple t_i of t_{i-1}, which is also a divisor of n, we call $(n;t_{i-1}:t_i)$-COALESCENCE the operation of combining (in the sense defined earlier) consecutive blocks of $m_i = t_i/t_{i-1}$ subsequences.

If we refer to the tree of Figure 1, we can easily see that each level of the tree corresponds to a coalescence of the input sequence. If we call COALESCER a network that performs a coalescence, we can build a COMBINATION-SORTER by cascading a suitable set of coalescers, as shown in Figure 5.

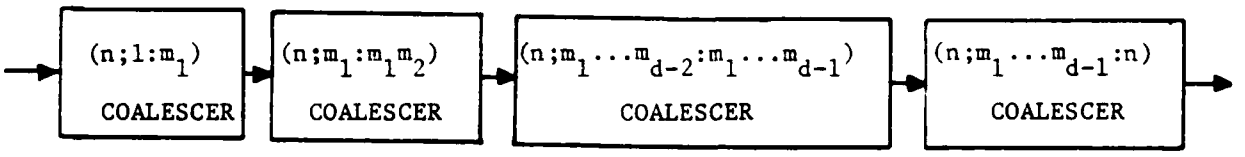


Figure 5. COMBINATION-SORTER as a cascade of COALESCERS.

5.1. The COALESCER

An $(n;t_{i-1} : t_i)$-COALESCER can be easily constructed by using $n_i \triangleq n/t_i$ (m_i,t_{i-1})-COMBINERS. Let us assume, for simplicity, that n_i is a perfect square. We can then lay out the combiners in a $\sqrt{n_i} \times \sqrt{n_i}$ array with input and output lines running in a chosen direction, say, parallel to the rows. An example with $n_i = 4$ is shown in Figure 6.

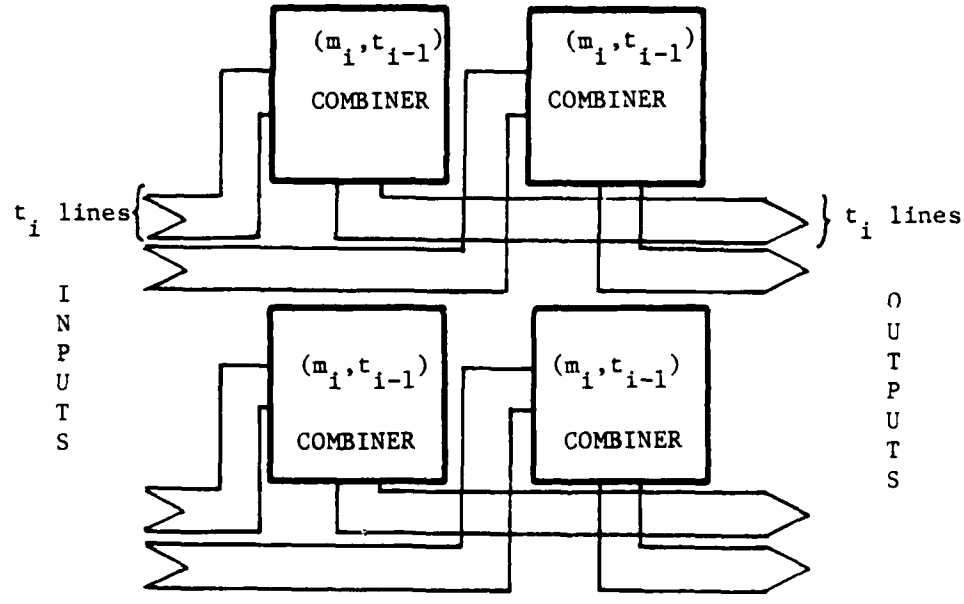


Figure 6. Layout of an $(n;t_{i-1} : t_i)$-COALESCER with $n_i = n/t_i$ $(m_i=t_i/t_{i-1},t_{i-1})$-COMBINERS.

We now estimate the area of the COALESCER. We first assume to use full-tree COMBINERS, so that the side of the COMBINER has a length of $O(t_i \log m_i)$. For the layout shown in Figure 6, we then have:

$$\text{height} = O(\sqrt{n_i}\, t_i \log m_i + n_i t_i) = O\left(n\left(1+\frac{\log m_i}{\sqrt{n_i}}\right)\right)$$

$$\text{width} = O(\sqrt{n_i}\, t_i \log m_i) = O\left(n\left(\frac{\log m_i}{\sqrt{n_i}}\right)\right)$$

If instead we use comb-tree COMBINERS, the size becomes

$$\text{height} = O(n)$$

$$\text{width} = O\left(n\,\frac{\log m_i}{\sqrt{n_i}}\right)$$

The computation time is $T_F = O(\tau+q+\log m_i)$ for the full-tree COALESCER, and $T_C = O(\tau+q+m_i)$ for the comb-tree COALESCER. When $q = \vartheta(\log n)$, then $T_F = O(\log n)$. If, in addition, $m_i = O(\log n)$, then $T_C = O(\log n)$.

5.2. An Optimal VLSI Sorter

From the previous considerations it is easy to see that we can obtain a VLSI implementation of COMBINATION-SORTERS by suitable use of COALESCERS. It should also be easy to compute time and area, once the factorization $n = m_1 m_2 \ldots m_d$ for the algorithm is chosen.

We now show that there is a COMBINATION-SORTER for words of length $q = \vartheta(\log n)$ that sorts n elements in time $T = O(\log n)$ and area $A = O(n^2)$, thus achieving the known lower bound for this problem. The sorter we propose is given by the block diagram in Figure 7.

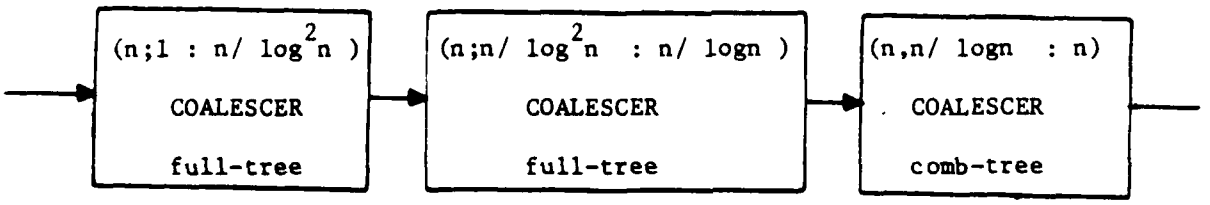


Figure 7. A COMBINATION-SORTER with three COALESCERS, for optimal VLSI sorting.

From the general analysis we easily see that the coalescers take area (width × height) $O(n) \times O(n)$, $O(n\log\log n/\sqrt{\log n}) \times O(n)$, and $O(n) \times O(n)$ respectively. It is also clear that the total time is $O(\log n)$, thus our claim is proved.

6. Area-Time Trade-Off

The COMBINATION-sorter proposed in this paper has optimal area among sorters that achieve minimum computation times. It is now interesting to ask whether we can trade time for area, and build a slower but smaller sorter with optimal $AT^2 = \theta(n^2\log^2 n)$.

Since, as we already recalled in the Introduction, area-time optimal circuits for sorting can be built when $T \in [\Omega(\log^3 n), O(\sqrt{n\log n})]$ (for a $(1+\varepsilon)\log n$ wordlength) the range of computation times for which no optimal circuits is known yet is $[\Omega(\log n), O(\log^3 n)]$.

We will now describe a network, which, by choosing an appropriate value for a design parameter, allows us to sort in $A = O(n^2\log^2 n/T^2)$ for any time $T \in [\Omega(\log n), O(\sqrt{n\log n})]$. The network is the cascade interconnection of two components. The first component is a COMBINATION-sorter for $\frac{n}{s}$ inputs. The second component is a new general architecture, called the mesh-of-CCC (MCCC) and obtained by suitably "hybridizing" known networks (the mesh and the CCC). This architecture will now be described in detail.

An (n,s)-MCCC, with $n = 2^\nu$, $s = 2^\sigma$, and $r \triangleq n/s^2 = 2^\rho$ $(\rho=\nu-2\sigma)$ consists of s^2 CCC modules, each with r cycles of length ρ. The $n \times \rho$ processing elements of the MCCC are conveniently indexed as

$$M_{ij}(h,k): \quad 0 \le i,j < s, \ 0 \le h < \rho, \ 0 \le k < r.$$

For a fixed (i,j) pair the set $\{M_{ij}(h,k): 0 \le h < \rho, 0 \le k < r\}$ is connected as a CCC-module, exactly as described in Section 3. Then CCC modules are arranged as an $s \times s$ mesh, and, for a fixed k, the set of micromodules $\{M_{ij}(0,k): 0 \le i,j < s\}$ is mesh-connected (with i and j as row and column indices respectively).

The MCCC closely resembles the COMBINER architecture defined in Section 4, and more specifically, the version with comb-trees. In fact the MCCC could be obtained from the comb-tree connected CCC's by identifying in all CCC's micromodules $M_{ij}(h,k)$ and $M_{ij}(h,k+t)$ (with $0 \leq k < t$), and deleting the edges related to E_τ.

The mesh of CCC's is a very interesting network in its own right, and we shall now show how it can: (i) emulate the ASCEND (or DESCEND) paradigm [10] of the Binary-Cube in optimal $AT^2 = O(n^2\log^2 n)$ for any computation time $T \in [\Omega(\log^2 n), O(\sqrt{n}\log n)]$; (ii) emulate the SORTING paradigm [3] in optimal $AT^2 = O(n^2\log^2 n)$ for any computation time $T \in [\Omega(\log^3 n), O(\sqrt{n}\log n)]$. (Recall that we are referring to a $\theta(\log n)$ input words, and to a word-serial mode of operation.)

If we consider a ν-dimensional binary cube whose processors are $P_0, P_1, \ldots, P_{n-1}$ $(n = 2^\nu)$, we can establish the following correspondence between MCCC micromodules, and cube processors:

$$M_{ij}(0,k) \leftrightarrow P_t, \quad t = \frac{n}{s} j + \frac{n}{s^2} i + k.$$

Then it is easy to see that dimension $E_0, \ldots, E_{\rho-1}$ of the cube are assigned to the CCC modules, dimensions $E_\rho, \ldots, E_{\rho+\sigma-1}$ are assigned to the mesh columns, and finally dimensions $E_{\rho+\sigma}, \ldots, E_{\rho+2\sigma-1}$ are assigned to the mesh rows. Thus, by application of well known techniques for emulating the cube with a CCC or a linear array [10], an ASCEND (or DESCEND) algorithm can be executed in $O(\rho+s)$ word-steps.

On the other hand the MCCC can be trivially laid out in an $O(n^2/s^2)$ square, since each CCC requires $O(\frac{n^2}{s^4})$ area and channels of $O(n^2/s^2)$ width allow a straightforward implementation of mesh-connections. In conclusion, for s in the range $[\Omega(\log n), O(\sqrt{n/\log n})]$, considering that $\rho = O(\log n)$ and that a word step takes $O(\log n)$ time, we obtain $T = O(s\log n)$ and $A = O(n^2/s^2)$, which gives an optimal AT^2.

The MCCC, used in the way just described, would not be optimal for the execution of bitonic sorting. Bitonic sorting of $n = 2^{\nu}$ elements consists of ν merging phases $M_0, M_1, \ldots, M_{\nu-1}$, with phase M_i performing the merging of pairs of sequences of length 2^i, and requiring on the cube the successive use of dimensions $E_{i-1}, E_{i-2}, \ldots, E_1, E_0$. So, the schedule of use of dimensions for a complete sorting is

$$\underbrace{E_0}_{M_0};\ \underbrace{E_1, E_0;}_{M_1}\ \underbrace{E_2, E_1, E_0;}_{M_2}\ \ldots,\ \underbrace{E_{\nu-1}, E_{\nu-2}, \ldots, E_0}_{M_{\nu-1}}$$

For brevity, we shall call this schedule [3] the sorting paradigm. On the MCCC the sorting paradigm requires $O(\rho \log n + s \log s)$ word steps, more than we desire.

We can eliminate the logs factor by a technique (already successful in the construction of the Pleated-CCC) consisting of an alternate arrangement of the 2σ topmost dimensions to columns and rows. More precisely, if

$$i = \sum_{\ell=0}^{\sigma-1} i_\ell 2^\ell, \quad j = \sum_{\ell=0}^{\sigma-1} j_\ell 2^\ell, \quad t' = \sum_{\ell=0}^{\sigma-1} (2i_\ell + j_\ell)\, 2^{2\ell},$$

then we establish between MCCC micromodules and cube processors the correspondence:

$$M_{ij}(0,k) \leftrightarrow P_t, \quad t = t' \frac{n}{s^2} + k.$$

For this correspondence a simple argument (similar to one given in [3]) shows that only $O(\rho \log n + s)$ word-steps are required for execution of the sorting paradigm.

Again, for $s \in [\Omega(\log^2 n), O(\sqrt{n/\log n})]$, recalling that $O(\log n)$ time is used for a word step, we obtain $T = O(s \log n)$, and $A = O(n^2/s^2)$. Although our main purpose in defining the MCCC is to construct optimal sorters for $T \in [\Omega(\log n), O(\log^3 n)]$, we have seen that the MCCC is an

optimal emulator of the cube for both the ASCEND and the SORTING paradigms. Let us also point out another interesting feature of the MCCC, namely that the maximum edge-length in the layout is $O(\frac{n}{s})$. For $s = \theta(\log n)$ we obtain a maximum (edge-length) $= O(n/\log^2 n)$, which is optimal.[3] In fact [12] maxedge-length $= \Omega(\sqrt{\text{optimal area}}/\text{diameter})$ for any graph, and for the MCCC optimal area $= \theta(n^2/\log^2 n)$, and diameter $= \theta(\log n)$. It is also interesting to recall that the optimal layouts known for the CCC and the Shuffle-Exchange contain edges of length $O(n/\log n)$.

To obtain networks faster than the MCCC we start from the following observation. A COMBINE-sorter with n/s input can sort (in time $O(s\log n)$ and area $O(n^2/s^2)$) s sequences of n/s elements each. These sequences can then be fed, say one per column, into an MCCC with parameter s. The sequence in each CCC module is at this point already sorted, and the MCCC is ready (after inverting the order of some sequences to comply with bitonic sorting rules) to execute the last 2σ merging phases. (For the sake of simplicity we will ignore the fact that only σ phases would be really necessary after the work done by the COMBINE-sorter.) A simple analysis allows us to conclude that, in the process, the MCCC executes $O(\log s+s)$ steps using $O(\log n)$ time each thus running for a total time $T = O(s\log n)$.

In conclusion, when $s \in [\Omega(1), O(\log^2 n)]$, the computation time T of the entire machine ranges in $[\Omega(\log n), O(\log^3 n)]$, and for each T the layout area is optimally $\theta(n^2 \log^2 n/T^2)$.

---

[3] This property has been noted also by A. Aggarwal for an architecture very similar to the MCCC (private communication).

REFERENCES

1. C. D. Thompson, "The VLSI complexity of sorting," to appear.

2. C. D. Thompson, A Complexity Theory for VLSI, Ph.D. Thesis, Computer Science Department, Carnegie-Mellon Univ., Aug. 1980.

3. G. Bilardi, F. P. Preparata, "A VLSI optimal architecture for bitonic sorting," Proc. 7th Conf. on Information Sciences and Systems, The Johns Hopkins University, Baltimore, MD, (March 1983); pp. 1-5.

4. K. E. Batcher, "Sorting networks and their applications," Proc. AFIPS Spring Joint Computer Conference, vol. 32, pp. 307-314, April 1968.

5. D. D. Nath, S. N. Maheshwari, and P. C. P. Bhatt, "Efficient VLSI networks for parallel processing based on orthogonal trees," IEEE Trans. Comp., vol. C-32, no. 6, pp. 569-581, June 1983.

6. F. T. Leighton, "New lower bound techniques for VLSI," Proc. 22nd Symp. on the Foundations of Computer Science, IEEE Computer Society, Oct. 1981.

7. D. E. Muller, F. P. Preparata, "Bounds to complexities of networks for sorting and for switching," JACM, vol. 22, pp. 195-201, April 1975.

8. F. P. Preparata, "New parallel sorting schemes," IEEE Trans. Comput., vol. C-27, no. 7, pp. 669-673, July 1978.

9. M. C. Pease, "The indirect binary n-cube microprocessor array," IEEE Trans. Comput., vol. C-26, no. 5, pp. 458-473, May 1977.

10. F. P. Preparata, J. Vuillemin, "The cube-connected-cycles: A versatile network for parallel computation," Com. of the ACM, vol. 24, no. 5, pp. 300-309, May 1981.

11. D. Nassimi, S. Sahni, "Parallel permutation and sorting algorithms and a new generalized connection network," JACM, vol. 29, no. 3, pp. 642-667, July 1982.

12. F. T. Leighton, Layouts for the Shuffle-Exchange Graph and Lower Bound Techniques, Ph.D. Thesis, Dept. of Mathematics, MIT, August 1981.

13. M. Aitai, J. Komlos, E. Szemeredi, "An O(nlogn) sorting network," Proc. 15th SIGACT, Boston, MA, April 1983, pp. 1-9.


DAT
ILM